ESTIMADOR DE DESCARGAS PLAY STORE

# PROYECTO MACHINE LEARNING

Javier Calderón García

# Pasos del proyecto

1

Obtención de la información

Revisión de las características de los datos obtenidos

2

Procesamiento

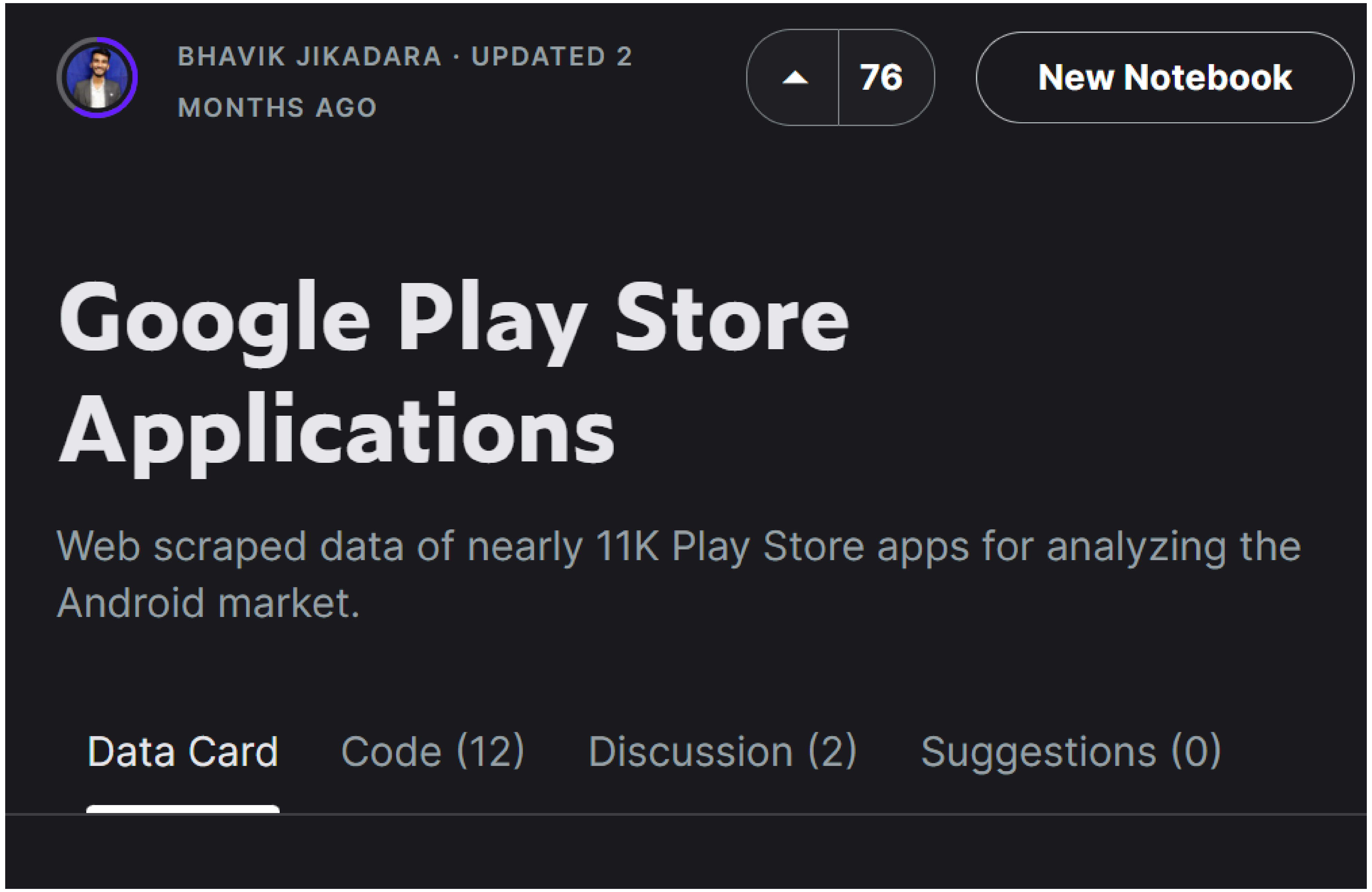
Feature Engineering

3

Entrenamiento de diferentes modelos.

4

Selección del modelo final.




# Selección de la información

Google Play Store Applications

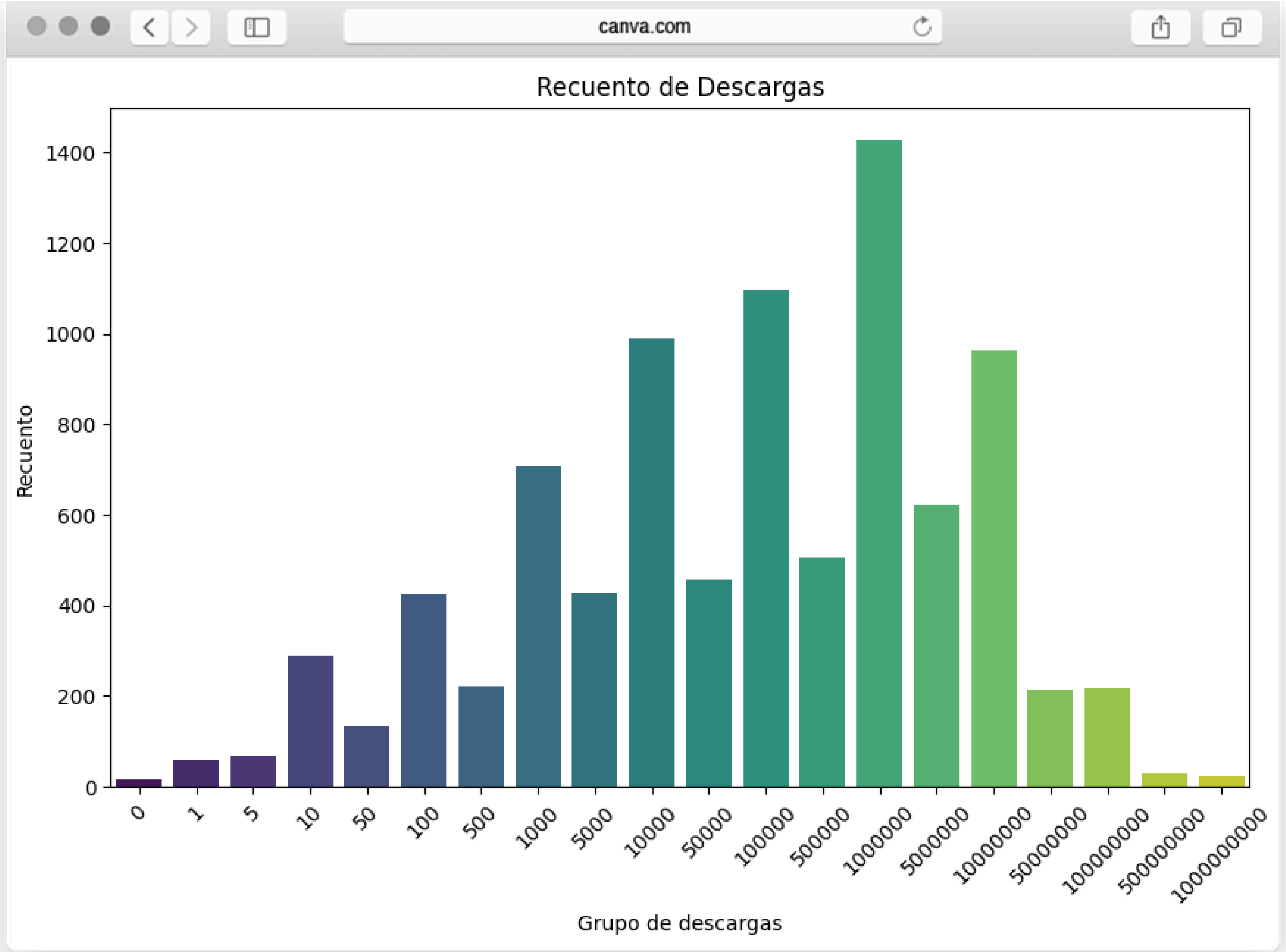
https://www.kaggle.com/datasets/bhavikjikadara/google-play-store-applications/data

# Revisión de los datos obtenidos

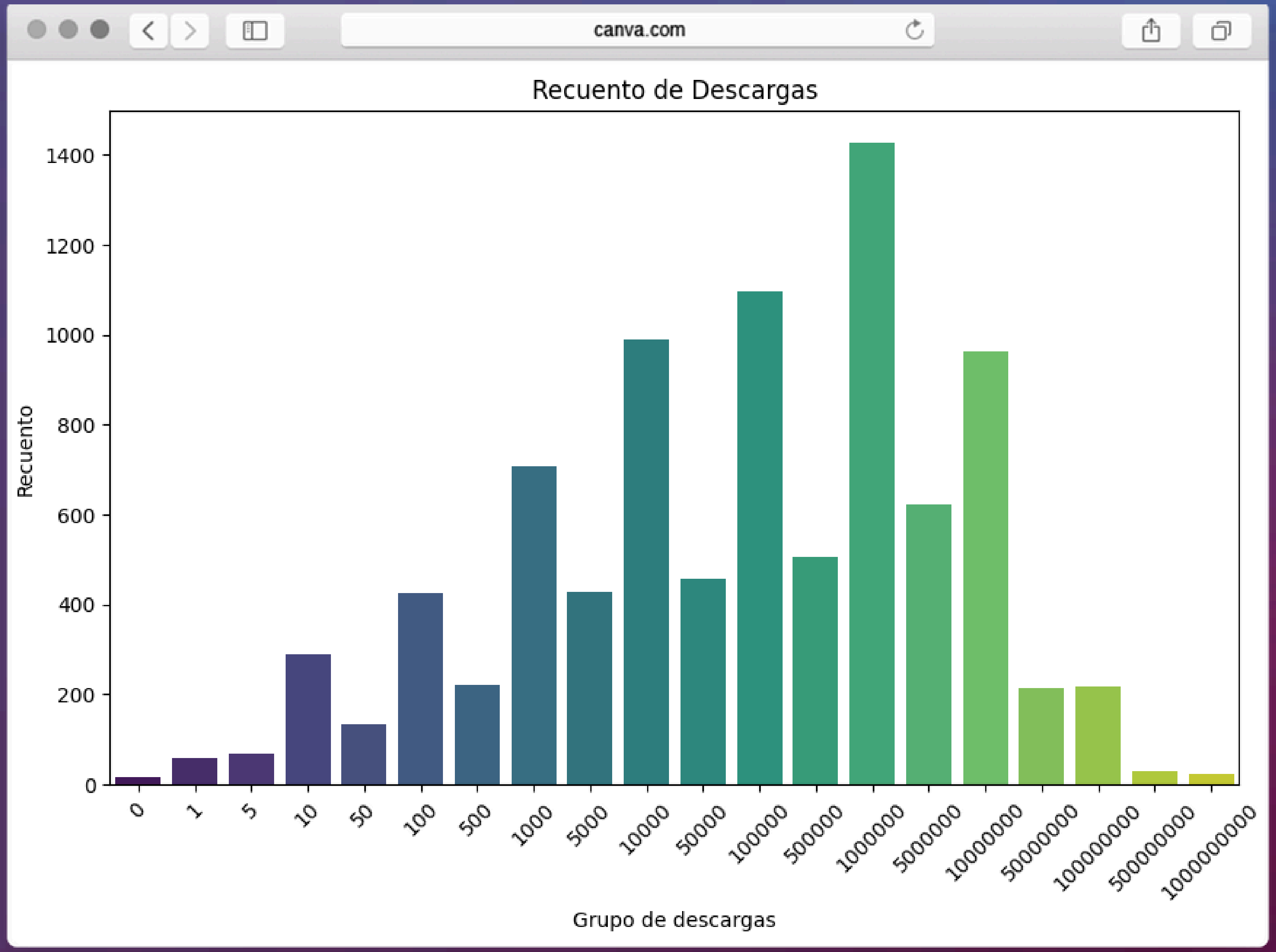
- De todas las variables del dataset, únicamente hay una continua.
- Disponemos de 10833 entradas, de las cuales 1074 están duplicados.

# Procesamiento

## Variable objetivo

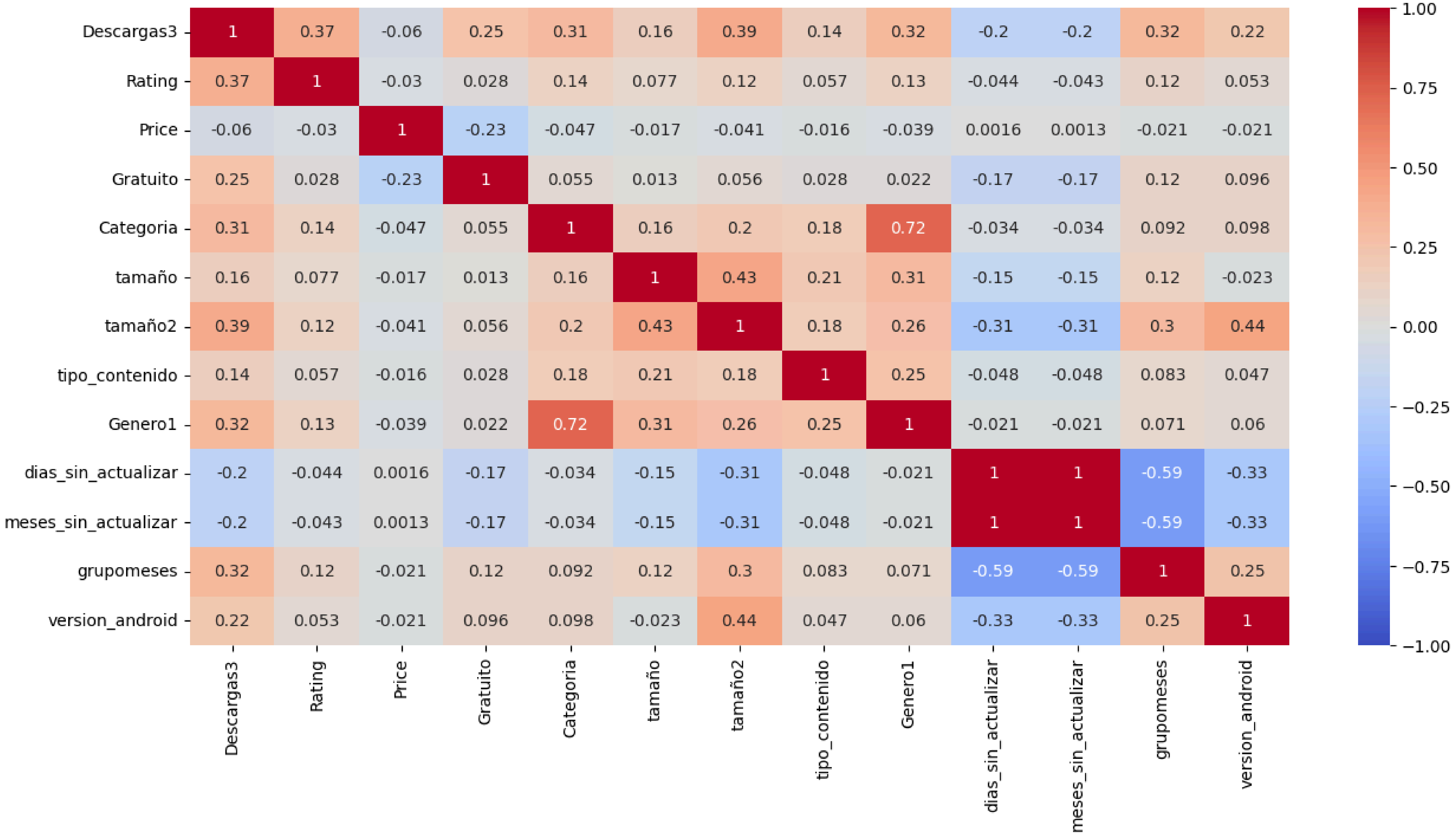



# Procesamiento
## Variable objetivo




# Procesamiento

## Resto de variables



# Feature Engineering

No es viable usar el número de reviews directamente, pero podemos usar su valor promedio, medianas o modas respecto al resto de variables.

## Grupomeses

Media de Reviews

```
for i,j in enumerate(list(train.groupby("grupomeses")["Reviews"].mean().values)):
    train.loc[train["grupomeses"]==i,"grupomeses_grp_Reviews_mean"]=j
```

```
test["grupomeses_grp_Reviews_mean"]=pd.merge(train, test, on="grupomeses", how="left")["grupomeses_grp_Reviews_mean"]
```

Mediana de Reviews

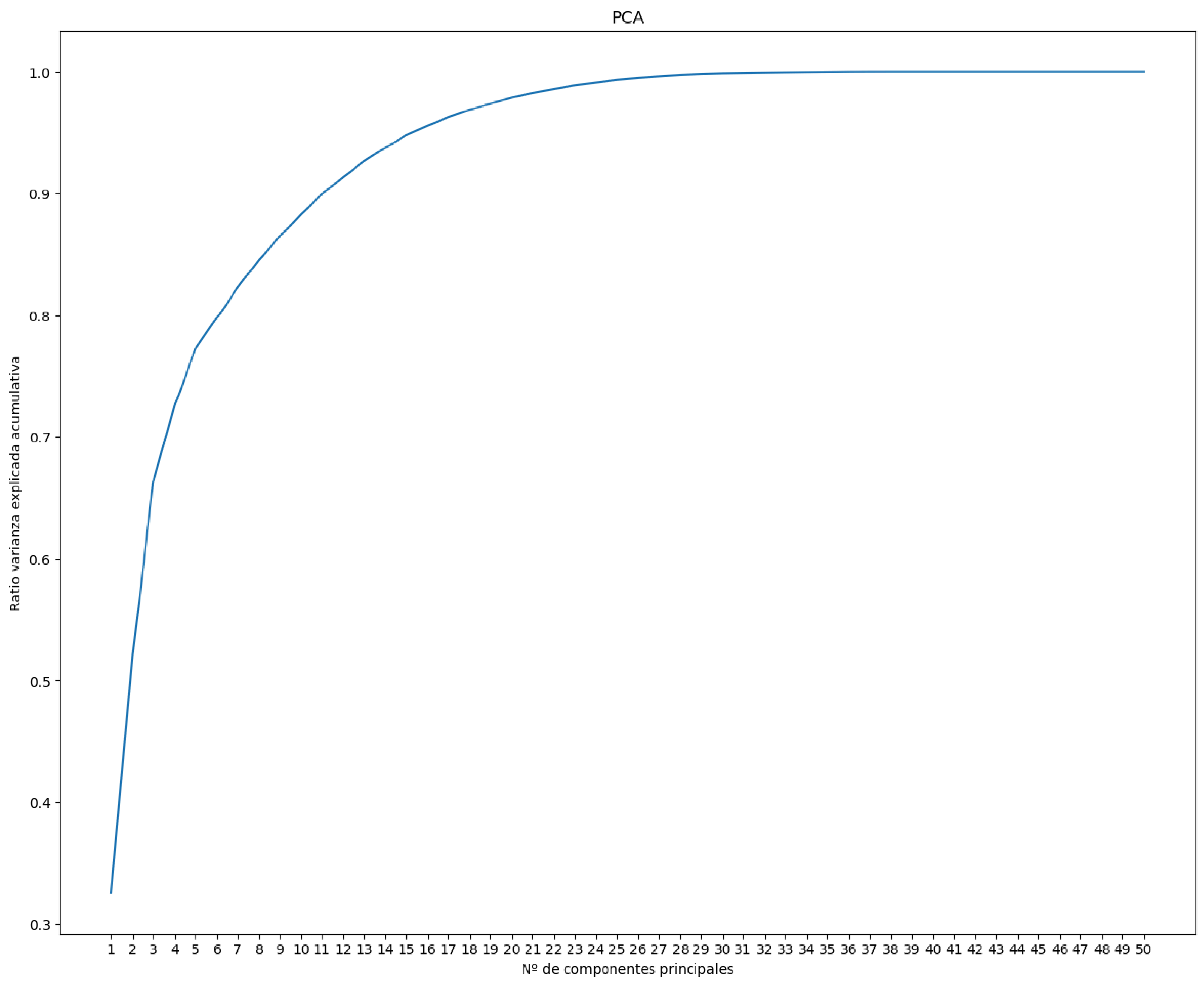
```
for i,j in enumerate(list(train.groupby("grupomeses")["Reviews"].median().sort_values().values)):
    train.loc[train["grupomeses"]==i,"grupomeses_grp_Reviews_median"]=j
```

```
test["grupomeses_grp_Reviews_median"]=pd.merge(train, test, on="grupomeses", how="left")["grupomeses_grp_Reviews_median"]
```

# Entrenamiento de modelos

## PCA

- Reducción de dimensionalidad.
- Eliminación de multicolinealidad



# Entrenamiento de modelos

## RandomForest

```
model = RandomForestClassifier(random_state=42)
model.fit(X_train,y_train)
predictions = model.predict(X_test)
print(model.score(X_train,y_train))
print("Accuracy",accuracy_score(y_test, predictions))
print("Recall",recall_score(y_test,predictions, average="macro"))
print("Conf. Matrix\n",confusion_matrix(y_test,predictions))
```

✓ 0.8s

```
0.9992963940193491
Accuracy 0.629395218002813
Recall 0.6212069252494637
Conf. Matrix
 [[226 120  43]
 [ 73 265 142]
 [ 18 131 404]]
```

```
model.fit(X,y)
predictions = model.predict(final_test_x)
print("Accuracy",accuracy_score(final_test_y, predictions))
```

✓ 0.9s

```
Accuracy 0.5514912774338773
```

```
model = RandomForestClassifier(random_state=42)
model.fit(X_train_resampled,y_train_resampled)
predictions = model.predict(X_test)
print(model.score(X_train_resampled,y_train_resampled))
print("Accuracy", accuracy_score(y_test, predictions))
print("Recall",recall_score(y_test,predictions, average="macro"))
print("Conf. Matrix\n",confusion_matrix(y_test,predictions))
```

✓ 0.6s

```
0.9993103448275862
Accuracy 0.6075949367088608
Recall 0.6078947154339266
Conf. Matrix
 [[254  97  38]
 [114 246 120]
 [ 35 154 364]]
```

```
model.fit(X_resampled,y_resampled)
predictions = model.predict(final_test_x)
print("Accuracy",accuracy_score(final_test_y, predictions))
```

✓ 0.7s

```
Accuracy 0.5475520540236354
```

# Entrenamiento de modelos

## Pipeline con varios modelos

```python
xgb_params = {
    'selectkbest__k': range(1, 52),
    "scaler": [StandardScaler(), MinMaxScaler()],
    'classifier': [XGBClassifier()],
    'classifier__n_estimators': randint(50, 500),
    "classifier__learning_rate": [0.001, 0.01, 0.1, 0.5, 1.0],
    "classifier__min_child_weight": randint(1, 11),
    "classifier__subsample": [0.5, 0.7, 0.9, 1.0],
    "classifier__colsample_bytree": [0.5, 0.7, 0.9, 1.0]

}

knn_params = {
    'selectkbest__k': range(1, 52),
    "scaler": [StandardScaler(), MinMaxScaler()],
    'classifier': [KNeighborsClassifier()],
    'classifier__n_neighbors':randint(1, 20),
    'classifier__weights': ['uniform', 'distance'],
    'classifier__p': [1, 2]
}


gb_params = {
    'selectkbest__k': range(1, 52),
    "scaler": [StandardScaler(), MinMaxScaler()],
    'classifier': [GradientBoostingClassifier()],
    'classifier__n_estimators': randint(50, 500),
    'classifier__learning_rate': [0.001, 0.01, 0.1, 0.5, 1.0],
    'classifier__min_samples_split': randint(2, 25),
    'classifier__min_samples_leaf': randint(1, 11),
    'classifier__subsample': [0.5, 0.7, 0.9, 1.0]
}
```

```
Resultados X e y resampleados:
0.638716938660664
0.6491520833989282
0.634849194039303
[[316 158  31]
 [ 99 340 154]
 [ 22 178 479]]
Resultados X e y completos:
0.6437816544738323
0.6666527105345961
0.6362667621273648
[[289 178  38]
 [ 66 362 165]
 [ 14 172 493]]
```

XGBClassifier

# Modelo Final

Pipeline con:

- StandarScaler
- SelectKBest(k=42)
- XGBClassifier:
    - Learning_rate = 0.01
    - Min_child_weight=13
    - N_stimators = 621

Model Score:  0.717
Accuracy Score:  0.653
Precision Score: 0.686
Recall Score:  0.642

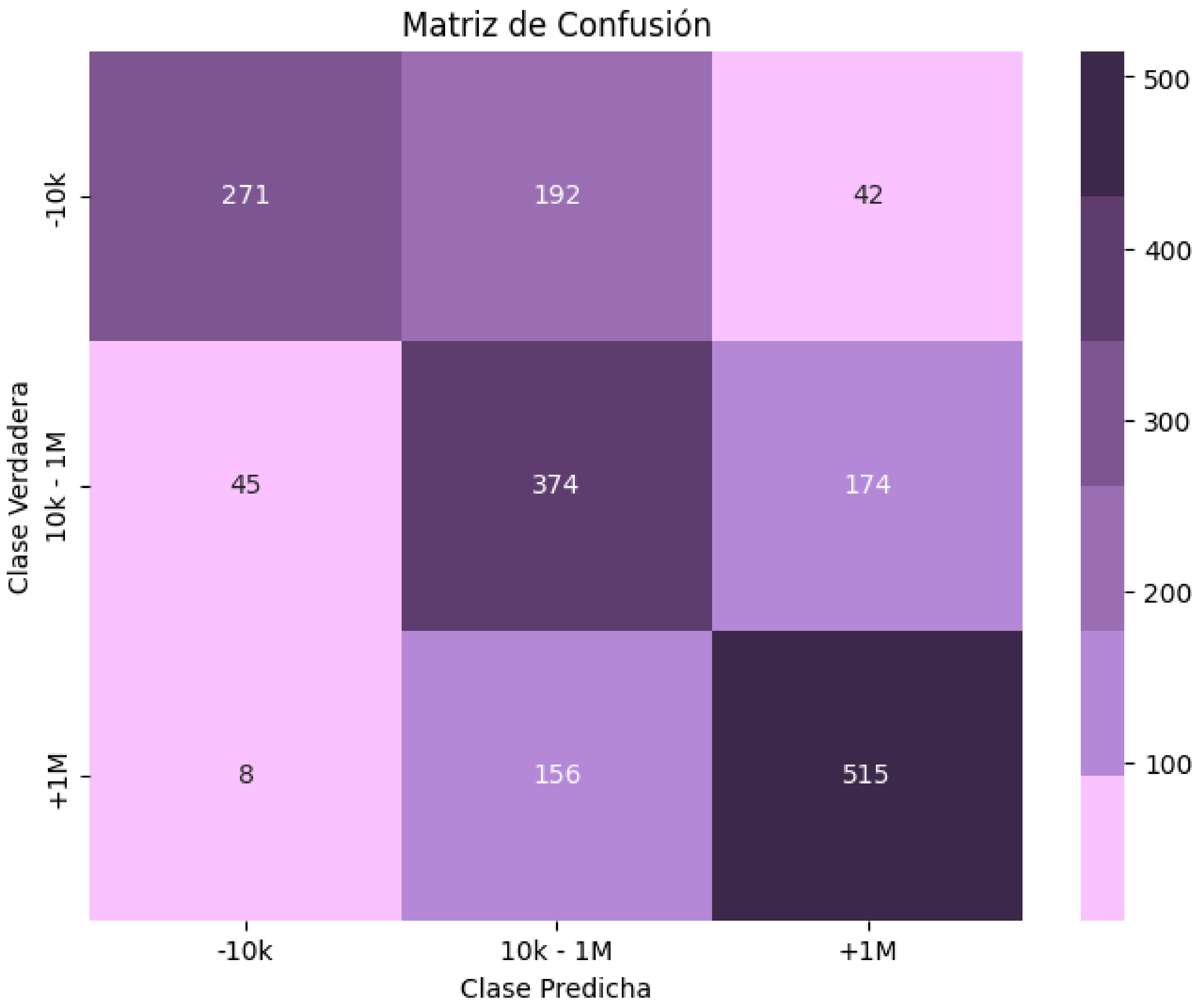
```
best_params = {
    'selectkbest__k': 42,
    'scaler': StandardScaler(),
    'classifier': XGBClassifier(learning_rate=0.01,
                                min_child_weight=13,
                                n_estimators=621,
                                )
}

# Construcción del pipeline con los mejores parámetros
best_pipeline = Pipeline(steps=[
    ("scaler", best_params['scaler']),
    ("selectkbest", SelectKBest(k=best_params['selectkbest__k'])),
    ("classifier", best_params['classifier'])
])
```

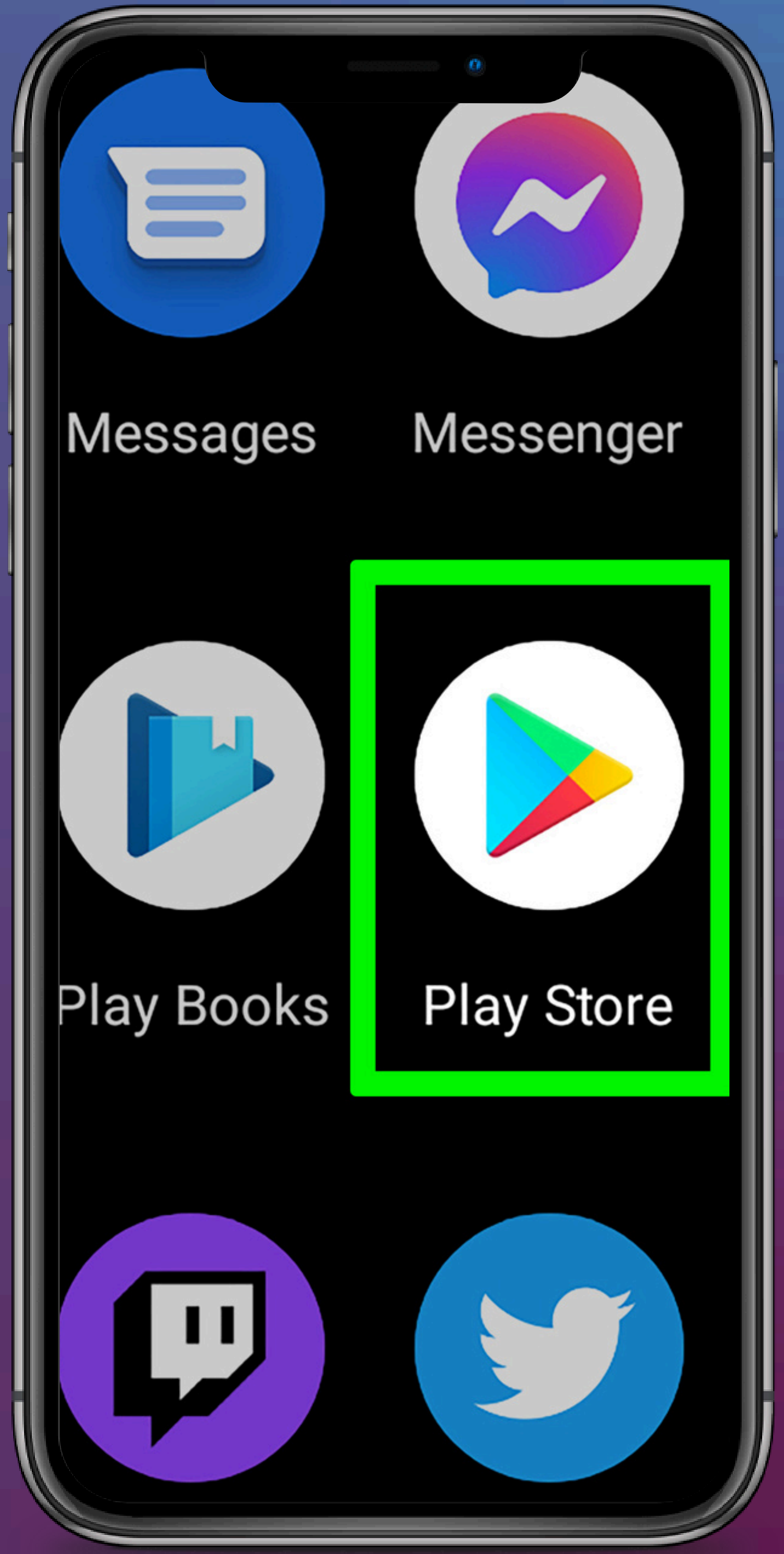
# Modelo Final

Pipeline con:

- StandarScaler
- SelectKBest(k=42)
- XGBClassifier:
  - Learning_rate = 0.01
  - Min_child_weight=13
  - N_stimators = 621

Model Score:  0.717
Accuracy Score:  0.653
Precision Score: 0.686
Recall Score:  0.642






# Conclusiones

El preprocesamiento ha jugado un papel fundamental a la hora de obtener un modelo viable.

El Rating, el precio, el tamaño y la fecha de la última actualización de la app son las variables más influyentes en el desarrollo de las predicciones.

Tras pruebas con varios modelos, el XGB es el que mejores resultados ha presentado mostrando un mejor desempeño a la hora de generalizar.